A BRANCH AND EXCLUDE ALGORITHM FOR THE KNAPSACK PROBLEM

BY

Robert Lawrence Hegerich

UNITED STATES
NAVAL POSTGRADUATE SCHOOL

THESIS

A BRANCH AND EXCLUDE ALGORITHM

FOR THE KNAPSACK PROBLEM

by

Robert Lawrence Hegerich

June 1968

A BRANCH AND EXCLUDE ALGORITHM

FOR THE KNAPSACK PROBLEM

by

Robert Lawrence Hegerich
Major, United States Army
B.S.E.E., Northeastern University, 1959

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 1968

ABSTRACT

A branch and exclude algorithm for solution of the "knapsack problem", max $\sum_{i=1}^{N} v_i x_i$ where $\sum_{i=1}^{N} w_i x_i \leq W$ and $x_i = 0,1$, is presented which requires relatively small amounts of computer running time and core storage allocation. In addition, a branch and bound scheme is developed. The branch and exclude method is then compared to the branch and bound method and to a branch and bound method given by Kolesar [2]. Computational results are given.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Stated in precise terms, the knapsack problem is one in which one wishes to select from among a finite collection of indivisible objects a subcollection which maximizes a linear function subject to a linear inequality constraint [2]. Dantzig [4] gives the following example: a person is planning a hike and has decided not to carry more than 70 pounds of different items, such as bed roll, geiger counter, cans of food, etc. We try to formulate this in mathematical terms. Let w_i be the weight of the i^{th} object and v_i be its relative value determined by the hiker in comparison with the other objects he would like to have on his trip. Let $x_i = 1$ mean that the i^{th} item is selected, and $x_i = 0$ mean that it is not selected. We express the weight limitation by $\sum_{i=1}^{N} w_i x_i \leq 70$, with $x_i = 0$ or 1, and we wish to choose the x_i so that the total value, $\sum_{i=1}^{N} v_i x_i$ is a maximum.

The knapsack problem has also been used as a model to formulate and solve various aspects of problems arising in capital investment [5], network reliability [6], capital budgeting [8], and in optimal methods of cutting stock [7]. Dantzig also uses this method in [4] to give a delightful "proof" that, of all the forms of marriage (monogamy, bigamy, polygamy) monogamy is the best of all possible relations.

The knapsack problem may be formulated as an integer programming problem and solved by any one of the following techniques: (i) the cutting plane technique, (ii) techniques

employing parallel shifts of the objective function hyperplane, (iii) techniques based upon Boolean algebra, and (iv) combinatorial methods which enumerate a restricted subset of possible integer solutions. For detailed references on these methods see Glover [1]. An alternative formulation of the problem may be developed through the use of dynamic programming [9]. The technique to be presented in this paper falls into the class of combinatorial methods mentioned in (iv) above.

In combinatorial methods of this type, the computer memory and running time requirements are usually quite large for problems involving many variables.

We present here first a branch and bound algorithm that is more in the spirit of Land and Doig [3] than is [2]. We then develop an algorithm which requires little computer storage and running. Computational results are then presented for our two algorithms and for Kolesar's method.

2. Formulation of the Problem

We formulate the knapsack problem as follows:

(1) Maximize $\sum_{i=1}^{N} v_i x_i$

Subject to $\sum_{i=1}^{N} w_i x_i \leq W,$

Where $x_i = 0, 1$ for $i=1,2,.....N$

We can assume without any loss in generality that the constants v_i and w_i are integers. Fractions can be handled by multiplying through by a proper factor. Non-positive constants may be handled by the following method which is due to Glover [1]:

i) Let $x_i = \begin{cases} 1 & \text{if } v_i \geq 0 \text{ and } w_i \leq 0 \\ 0 & \text{if } v_i \leq 0 \text{ and } w_i \geq 0 \end{cases}$

ii) Next reformulate the problem as:

Maximize $\sum_{i=1}^{N} v_i y_i$

Subject to $\sum_{i=1}^{N} w_i y_i \leq W$,

Where $y_k = 0,1$ for $i=1,2,.....N,$

and where $y_i = \begin{cases} x_i & \text{if } v_i, w_i \geq 0 \\ 1-x_i & \text{if } v_i, w_i \leq 0 \end{cases}$

The following example should illustrate this technique:

Maximize $3x_1 - 2x_2 - 4x_3 + x_4$

Subject to $2x_1 - 3x_2 + x_3 - x_4 \leq 0$.

Using Glover's procedure this problem is reformulated as:

$$\text{Maximize} \quad 3y_1 + 2y_2 - 1$$

$$\text{Subject to } 2y_1 + 3y_2 \leq 4$$

and the optimal solution if the integer constraint on x_2 is relaxed is 10/3 units with X = (1,1/3,0,1).

3. A Branch and Bound Algorithm

As shown by Dantzig [4] the optimal fractional solution to

$$(2)\quad \text{Maximize} \quad \sum_{i=1}^{N} v_i x_i$$

$$\text{Subject to} \quad \sum_{i=1}^{N} w_i x_i \leq W$$

$$0 \leq x_i \leq 1 \quad (i=1,\ldots,N)$$

is given by

$$x_i = 1 \quad \text{if } i < r$$

$$x_i = 0 \quad \text{if } 1 > r$$

$$x_r = \frac{W - \sum_{i<r} w_i}{w_r}$$

where r is the least integer ($0 \leq r \leq N$) for which $\sum_{i \leq r} w_i \geq W$. If no r exists we have all $x_i = 1$. If $x_r = 0$, we have the optimal solution to (1).

If x_r is fractional the value of the objective function is $z(1) = \sum_{i<r} v_i + v_r x_r$. We consider z(n), the value of the objective function at node n, as the solution to (2) with assigned variables added as constraints. A branch and bound scheme is essentially as follows:

1. Label node 1 with z(1). Go to 2.

2(a). Find the terminal node with the largest value of z(n). This is the node at which the next branching will take place. Any node ($\neq 1$) contains the effect of assigning values to variables and solving (2) with the assigned values of the variables added as constraints.

(b). If the solution at node n has all integer variables, we have achieved an optimal solution to (1). Stop. If not, go to 3.

3(a). Set n = n+1 and some unassigned variable, say x_t, equal to 0. Solve (2) with all assigned variables added as constraints. Label node n with the value z(n). Go to 3(b).

(b). Set n = n+1 and $x_t = 1$. Solve (2) with all assigned variables added as constraints. Label node n with the value z(n). Go to 2.

Kolesar's algorithm consists in taking as x_t, in step 3(a), the unassigned variable with smallest index (i.e., t = i for which v_i/w_i is the maximum for unassigned variables x_i).

We propose instead to follow [3] and to take x_t, in step 3(a), the variable that is fractional at node n.

To illustrate our branch and bound algorithm, we solve the problem given in [2]:

Index	w_i	v_i
1	30	60
2	50	60
3	40	40
4	10	10
5	40	20
6	30	10
7	10	3

and W = 100. Solving (2), we obtain z(1) = 140 with $x_1 = 1$, $x_2 = 1$, $x_3 = 1/2$. We label node 1 with z(1) = 140. Since x_3 is fractional we branch from node 1 and proceed to step 3(a). We solve (2) with $x_3 = 0$. This produces z(2) = 135 with $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$, $x_5 = 1/4$. We label node

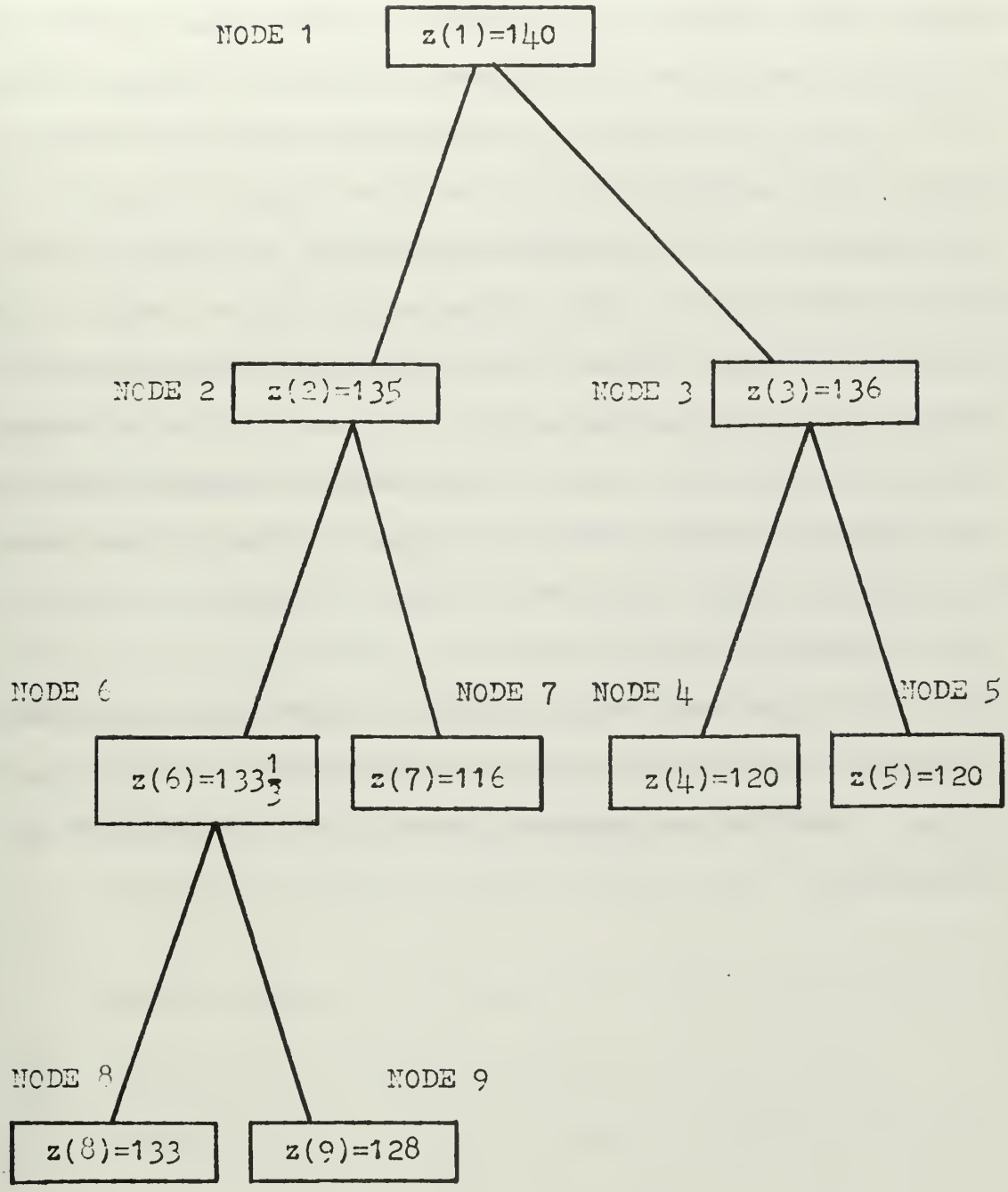


Figure 1. Complete tree for the example

2 with $z(2) = 135$. In step 3(b) we set $x_3 = 1$. This produces $z(3) = 136$ with $x_1 = 1$, $x_2 = 3/5$, $x_3 = 1$. We go to step 2 in the algorithm. We see that $z(3)$ is the maximum for all terminal nodes. Since the solution at node 3 has x_2 fractional we branch from node 3 and proceed to 3(a). We solve (2) with $x_3 = 1$ and $x_2 = 0$ added as constraints. This produces $z(4) = 120$, with $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$, $x_5 = 1/2$. We proceed to 3(b) and solve (2) with $x_3 = 1$ and $x_2 = 1$. This produces $z(5) = 120$ with $x_1 = 1/3$, $x_2 = 1$, $x_3 = 1$. Returning to step 2 we see that $z(2)$ is the maximum for all terminal nodes. Since the solution at node (2) has x_5 fractional we must branch from node 2. The method continues easily and the complete tree is shown in Figure 1. The optimal solution is given at node 8 with $z(8) = 133$, $x_1 = 1$, $x_2 = 1$, $x_4 = 1$, $x_7 = 1$ and all other $x_i = 0$.

The algorithm achieved solution with the generation of 9 nodes. The same problem was solved in [2] and required 15 nodes. Further computational experience is presented in Section 5.

4. A Branch and Exclude Algorithm

Branch and bound techniques normally require large amounts of computer storage because of the necessity to store information for all terminal nodes. In this section, we present a branch and exclude algorithm for solving (1) that requires little computer storage.

Our branch and exclude algorithm first finds an obvious integer solution to the constraints of (1). This solution is a lower bound to the optimal solution. We develop a branch of a tree and explore each part of the branch until the lower bound is reached or until a new feasible solution is found that represents a larger lower bound. We then back-track and develop new branches of the tree developing possibly larger lower bounds. Further branching is excluded when the lower-bound is reached. The algorithm stops when all new branches are excluded. The only information that is stored is the current lower bound solution and the branch routing. At the end, the lower bound solution is optimal.

Define [x] as the greatest integer less than or equal to x.

Define $S(x_0,x_1,\ldots,x_N)$ as the current lower bound solution, where the x_i are all given and $x_0 = \sum_{i=1}^{N} v_i x_i$.

Define $X(x_1,x_2,\ldots,x_N)$ to indicate assigned variables. A value $x_i = 2$ (or any number not equal to zero or one) indicates the variable is unassigned. An assigned variable will have the value zero or one.

Define $R(\mathbf{L})$ as the index of the L^{th} assigned variable.

The solution to (2), with L assigned components of X added as constraints, is

$$x_i = 1 \quad \text{if} \quad i < r,\ i \neq R(j) \quad (j = 1,\ldots,L)$$

$$x_i = 0 \quad \text{if} \quad i > r,\ i \neq R(j) \quad (j = 1,\ldots,L)$$

(3) $x_{R(j)} = 0$ or 1 $(j = 1,\ldots,L)$ depending on the assignment

$$x_r = (W - \sum_{i=1}^{L} w_{R(i)} x_{R(i)} - \sum_{i\varepsilon M(L)} w_i)/w_r$$

$$Z(L+1) = \sum_{i=1}^{L} v_{R(i)} x_{R(i)} + \sum_{i\varepsilon M(L)} v_i + v_r x_r$$

where the set M is given by

$$M(L) = \{i \mid i < r,\ i \neq R(j)\ (j = 1,\ldots,L)\}$$

and r is the least integer $(0 \leq r \leq N)$ for which

$$\sum_{i\varepsilon M(L)}^{L} w_i + w_r \geq W - \sum_{i=1}^{L} w_{R(i)} x_{R(i)} \quad .$$

If no r exists we have all $x_i = 1$ for $i \neq R(j)$ $(j = 1,\ldots,L)$. A lower bound to the solution of (1) is given by

$$x(L) = \sum_{i=1}^{L} v_{R(i)} x_{R(i)} + \sum_{i\varepsilon M(L)} v_i \quad .$$

The algorithm follows:

1. Set L = 1 and all components of X to two. Go to 2.
2. Solve (2). If the solution is all integer we have the optimal solution. Stop. If not the solution is $x_1 = 1$, $x_2 = 1,\ldots,x_{r-1} = 1$ and x_r is fractional. We calculate $x_0 = \sum_{i<r} v_i$ and form $S(x_0,1,1,\ldots,1,0,0,0)$ as a lower bound to the optimal solution, where the zero components in S represent $x_i = 0$, $i \geq r$. Set R(1) = r and the r^{th} component of X to zero. Go to 3.

3(a). Solve (2) with the L assigned components of X added as constraints.

We obtain (3). If $[Z(L + 1)] \leq x_0$ to to 4. If $Z(L + 1) > x_0$ and we have an integer solution, take $x_0 = Z(L + 1)$, form a new $S(x_0,x_1,x_2,\ldots,x_N)$ from (3), and go to 4. If $[Z(L + 1)] > x_0$ and we do not have an integer solution, go to 3(b).

(b). If $x(L) > x_0$, take $x_0 = x(L)$ and form a new $S(x_0,x_1,\ldots,x_N)$ from (3) with $x_r = 0$. In any case, Set $L = L + 1$, take $R(L) = r$, and set the R(L) component of X to zero. Go to 3(a).

4(a). If the R(L) component of X is equal to zero change the component to one and go to 3(a). If the R(L) component of X is one go to 4(b).

(b). If $L = 1$ the optimal solution is $S(x_0,x_1,\ldots,x_N)$. Stop. If $L \neq 1$ change the R(L) component of X to two, set $L = L-1$, and go to 4(a).

This completes the algorithm, z(L) represents the value of the objective function at the L^{th} level of a branch. R(L) represents a routing of assignments along the branch. Only one branch is studied at a time with preference given to assigning the value of zero to the fractional variable. This allows lower-bounds to be achieved more rapidly. The only permanent storage information required is the current lower bound solution S, the assigned variables R(L) at level L of the branch, and the assignment vector X.

In the problem given above the algorithm as viewed in Figure 1, goes as follows:

1. At node one the optimal solution to (2) is fractional. $x_0 = 120$.
2. Branch to node two, assigning $x_3 = 0$.
3. Branch to node six, assigning $x_5 = 0$.
4. Branch to node eight, assigning $x_6 = 0$. This produces an all integer result with $x_0 = 133$, $x_1 = 1$, $x_2 = 1$, $x_4 = 1$, $x_7 = 1$.
5. Back-track to node six; branch to node 9, assigning $x_6 = 1$.
6. Back-track to node 6, removing the assignment on x_6.
7. Back-track to node 2; branch to node 7, assigning $x_5 = 1$.
8. Back-track to node 2, removing the assignment on x_5.
9. Back-track to node 1; branch to node 3, assigning $x_3 = 1$. Calculate x(1) = 100; do not change the current lower bound.
10. Branch to node 4, assigning $x_2 = 0$.
11. Back-track to node 3; branch to node 5, assigning $x_2 = 1$.
12. Back-track to node 3, removing the assignment on x_2.
13. Back-track to node 1; end.
14. The optimal solution is given at node 8.

In this example, the total tree investigated has the same number of nodes.

5. Computational Results

We have programmed the two branch and bound schemes and the branch and exclude algorithm in Fortran IV and have run test problems on the IBM 360/67. A program for the branch and exclude method is given in Appendix A.

The coefficients v_i and w_i were generated as random integers and the three methods were then tried on the same problem. If

BK = Kolesar's branch and bound algorithm,

BB = the branch and bound algorithm presented here,

BE = the branch and exclude algorithm presented here,

we obtained the following results for the number of nodes generated and the time to achieve solution:

TABLE 1. Comparison of Methods

W	N	20	30	40	50	20	30	40	50
	BE	28.8	28.0	28.8	35.7	61	87	120	164
25	BB	26.4	20.6	26.6	28.6	201	282	395	494
	BK	56.2	49.6	48.0	50.0	259	345	447	552
	BE	34.5	38.7	36.1	54.5	75	111	145	235
50	BB	38.6	48.2	42.0	51.8	236	361	453	581
	BK	62.2	78.4	78.0	92.6	276	434	553	723
	BE	19.8	39.0	27.8	18.5	58	116	128	135
75	BB	31.0	30.2	27.2	25.0	222	312	404	485
	BK	62.6	101.2	84.4	61.6	280	523	578	616
	BE	7.2	24.3	46.8	29.0	36	97	181	163
100	BB	16.6	24.8	64.0	50.4	191	301	560	598
	BK	56.6	65.4	136.6	134.2	273	403	806	978

Average number of nodes generated. Average time in Milliseconds. (10 problems run to obtain each average)

In these test problems superiority of BB and BE over BK is clearly apparent with regard to number of nodes generated. As for time considerations, the branch and excludes method shows to be markedly faster over the other two methods, while BB is faster than BK. Various other problems than those listed above were run, and in general, the BE method achieved solution in about one-half to one-third the time required by the BK and BB methods.

The branch and exclude method was tried on several problems with 5000 variables and solution was achieved in all cases in approximately 4 minutes. The other two methods failed because of storage limitations. In general, the branch and exclude algorithm presented here can rapidly solve extremely large problems.

BIBLIOGRAPHY

1. Glover, F., "A Multiphase Dual Algorithm for the Zero-one Integer Programming Problem", Operations Research, Vol. 13 (1965), pp. 879-919.

2. Kolesar, P. J., "A Branch and Bound Algorithm for the Knapsack Problem", Management Science, Vol, 13 (1967), pp. 723-735.

3. Land, A. H., and Doig, A. G., "An Automatic Method of Solving Discrete Programming Problems", Econometrica, Vol. 28 (1960), pp. 497-520.

4. Dantzig, G. B., "Discrete Variable Extremum Problems", Operations Research, Vol. 5 (1957), pp. 256-277.

5. Hansmann, F., "Operations Research in the National Planning of Underdeveloped Countries", Operations Research, Vol. 9 (1961), pp. 203-248.

6. Kolesar, P., "Assignment of Optimal Redundancy in Systems Subject to Failure", Columbia University, Operations Research Group, Technical Report, 1966.

7. Gilmore, P. C., and Gomory, R. E., "A Linear Programming Approach to the Cutting Stock Problem-Part II", Operations Research, Vol. 11 (1963), pp. 863-888.

8. Cord, J., "A Method for Allocating Funds to Investment Projects when Returns are Subject to Undertainty", Management Science, Vol. 10, No. 2 (1964), pp. 335-341.

9. Bellman, R. E., and Dreyfus, S. E., Applied Dynamic Programming, Princeton University Press, 1962, pp. 27-31.

APPENDIX I

FORTRAN IV Computer Program for the Branch and Exclude Method

```
LEVEL 1, MOD 0              MAIN                    DATE = 68051

      IMPLICIT REAL(Y-Z,$),INTEGER(A-X)
      INTEGER*2 V,W,ORDER,X,SVECT,R,F,G,H
      COMMON V(100),W(100),N,X(100),ORDER(100),WNOT,
     *YVEC(100),SFLAG,XNOT,SVECT(100),L,R(100),F(100),
     *G(100)
      DIMENSION H(100)
C
C V,W,R,X AND L ARE AS STATED IN THE TEXTUAL EXPLANATION OF
C     THIS ALGORITHM
C WNOT IS THE WEIGHT CONSTRAINT, W
C N IS THE NUMBER OF VARIABLES
C SVECT IS THE SOLUTION VECTOR, S, EXCEPT THAT X-SUB-ZERO
C     IS KEPT AS XNOT
C SFLAG IS THE NUMBER ASSIGNED TO A PARTICULAR NODE
C
      READ 101,WNOT,N
101   FORMAT(2I5)
      DO 102 I=1,N
      READ 101,V(I),W(I)
102   CONTINUE
      DO 103 I=1,N
      R(I)=0
      X(I)=2
      SVECT(I)=0
103   CONTINUE
      XNOT=0
      SFLAG=0
      L=1
C
C THE VARIABLES ARE HERE PLACED IN ORDER OF DESCENDING
C    VSUBI/WSUBI
C
      CALL RANK
C
C THE OPTIMAL SOLUTION IS NOW FOUND BY THE SOLVIT ROUTINE
C
      CALL SOLVIT
C
C AT THIS POINT THE VARIABLES ARE REARRANGED INTO THEIR
C    ORIGINAL ORDER, AND THE RESULTS PRINTED
C
      DO 91 I=1,N
      F(ORDER(I))=SVECT(I)
      G(ORDER(I))=V(I)
      H(ORDER(I))=W(I)
91    CONTINUE
      PRINT 70,XNOT
70    FORMAT(1H1,20X,'SOLUTION =',I5,' UNITS')
      PRINT 90
90    FORMAT(  20X,'I',19X,'X(I)',16X,'V(I)',16X,'W(I)')
      DO 92 I=1,N
      PRINT 93,I,F(I),G(I),H(I)
93    FORMAT(3X,4(17X,I3))
92    CONTINUE
      STOP
      END
```

```
      SUBROUTINE RANK
      IMPLICIT REAL(Y-Z,$),INTEGER(A-X)
      INTEGER*2 V,W,ORDER,X,SVECT,R,F,G,H
      COMMON V(100),W(100),N,X(100),ORDER(100),WNOT,
     *YVEC(100),SFLAG,XNOT,SVECT(100),L,R(100),F(100),
     *G(100)
C
C  ORDER IS THE VECTOR WHICH WILL CONTAIN THE ORDER IN WHICH
C     THE VARIABLES ARE REARRANGED
C
      DO 1 I=1,N
      ORDER(I)=I
      Y=W(I)
      Z=V(I)
      YVEC(I)=Z/Y
1     CONTINUE
C
C  THE RANK ORDER IS NOW DETERMINED AND STORED IN ORDER
C
      K=0
2     K=K+1
      J=K+1
      ZAP=YVEC(K)
      M=K
      DO 3 I=J,N
      IF(YVEC(I).LE.ZAP)GO TO 3
      ZAP=YVEC(I)
      M=I
3     CONTINUE
      YVEC(M)=YVEC(K)
      YVEC(K)=ZAP
      PIP=ORDER(M)
      ORDER(M)=ORDER(K)
      ORDER(K)=PIP
      IF(J.NE.N)GO TO 2
C
C  THE VARIABLES ARE NOW REARRANGED
C
      DO 4 I=1,N
      F(I)=V(I)
      G(I)=W(I)
4     CONTINUE
      DO 5 I=1,N
      V(I)=F(ORDER(I))
      W(I)=G(ORDER(I))
5     CONTINUE
      RETURN
      END



      SUBROUTINE SOLVIT
      IMPLICIT REAL(Y-Z,$),INTEGER(A-X)
      INTEGER*2 V,W,ORDER,X,SVECT,R,F,G,H
      COMMON V(100),W(100),N,X(100),ORDER(100),WNOT,
     *YVEC(100),SFLAG,XNOT,SVECT(100),L,R(100),F(100),
     *G(100)
C
C  SUM IS THE VALUE OF THE SUM OF WSUBI*XSUBI FOR ALL I
C  TOTALIS THE VALUE OF THE SUM OF VSUBI*XSUBI FOR ALL I
C  YVEC IS A VECTOR WHICH HOLDS THE VALUE OF THE VARIABLES
C     AS EACH NODE IS GENERATED
C
```

```
10      DO 11 I=1,N
        YVEC(I)=0
11      CONTINUE
3       SUM=0
        TOTAL=0
        SFLAG=SFLAG+1
C
C  THE VARIABLES WHICH HAVE BEEN CONSTRAINED TO BE ONE ARE
C     HERE CONSIDERED
C
        DO 12 I=1,N
        IF(X(I).NE.1)GO TO 12
        YVEC(I)=1
        SUM=SUM+W(I)
        TOTAL=TOTAL+V(I)
        IF(SUM-WNOT)12,14,20
12      CONTINUE
C
C  IF THE CONSTRAINT EQUATION HAS NOT BEEN VIOLATED,
C     UNASSIGNED VARIABLES ARE NOW CONSIDERED
C
        DO 13 I=1,N
        IF(X(I).LE.1)GO TO 13
        YVEC(I)=1
C
C  THE INITIAL LOWER BOUND VECTOR,S, IS BEING FORMED BY THE
C     NEXT CARD
C
        IF(SFLAG.EQ.1)SVECT(I)=1
        SUM=SUM+W(I)
        TOTAL=TOTAL+V(I)
        IF(SUM-WNOT)13,14,15
13      CONTINUE
C
C  IF STATEMENT 14 IS REACHED, AN INTEGER SOLUTION HAS BEEN
C     FOUND
C
14      CALL STORE(TOTAL)
        IF(SFLAG.EQ.1)RETURN
C
C  THE VALUE ASSIGNED TO THE FRACTIONAL VARIABLE IS NOW
C     CHECKED, AND BRANCHING INSTRUCTIONS GENERATED
C
16      Q=X(R(L))
        IF(Q.EQ.0)GO TO 17
20      X(R(L))=2
        L=L-1
        IF(L.EQ.1)RETURN
        GO TO 16
17      X(R(L))=1
        GO TO 10
C
C  IF STATEMENT 15 IS REACHED, A NON-INTEGER SOLUTION HAS
C     BEEN FOUND
C
15      SUM=SUM-W(I)
        TOTAL=TOTAL-V(I)
        YVEC(I)=0
C
C  THE NEXT CARD CHECKS TO DETERMINE IF A NEW LOWER BOUND
C     CAN BE ENTERED
C
        IF(X(R(L)).EQ.1.AND.TOTAL.GT.XNOT)CALL STORE(TOTAL)
C
C  FORMULATION OF THE INITIAL LOWER BOUND IS COMPLETED BY
C     THE NEXT THREE CARDS
C
```

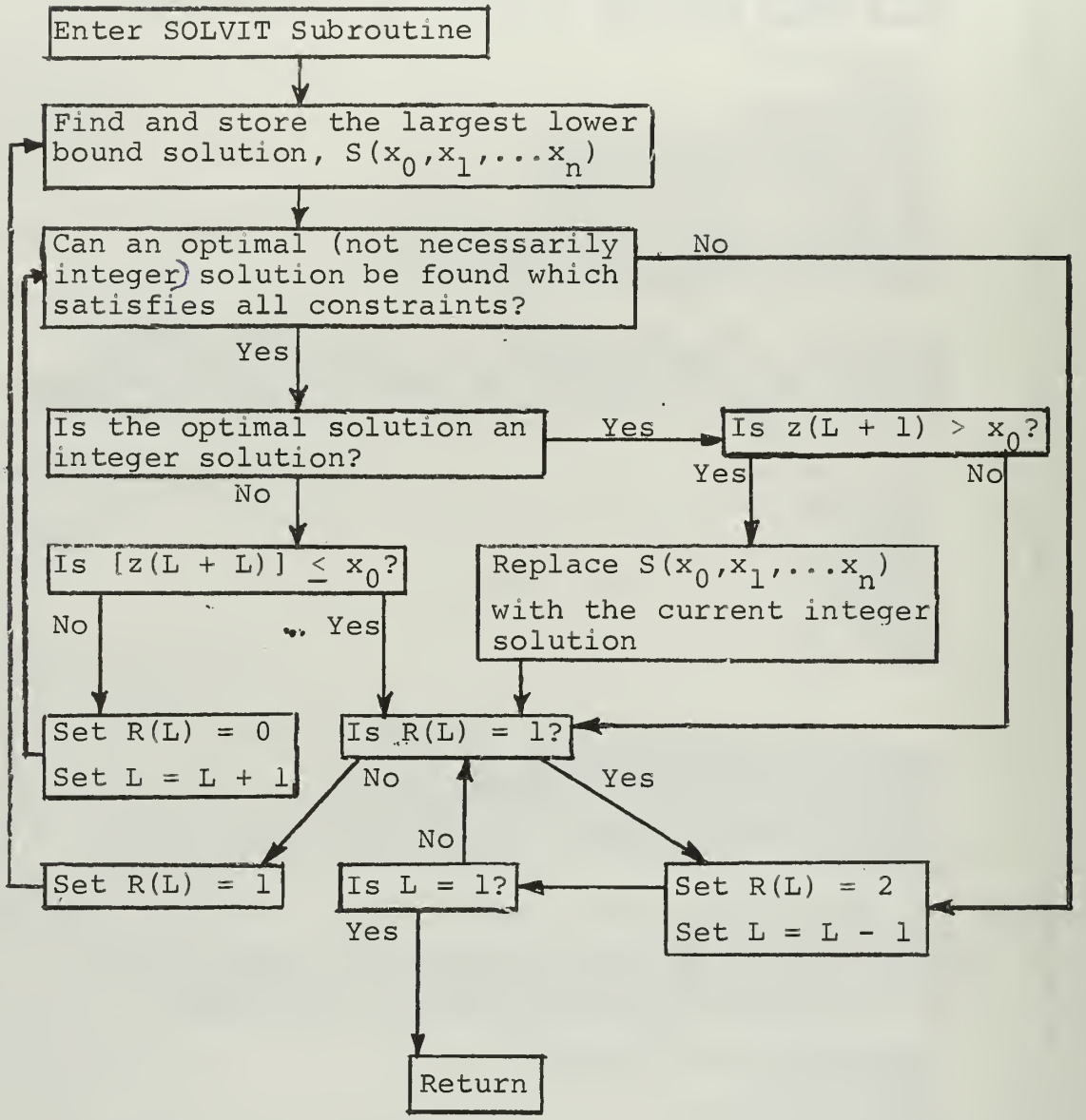
```
      IF(SFLAG.NE.1)GO TO 79
      SVECT(I)=0
      XNOT=TOTAL
C
C  A DETERMINATION IS NOW MADE AS TO WHETHER THE GREATEST
C     INTEGER CONTAINED IN THE VALUE OF THE PRESENT
C     NON-INTEGER SOLUTION IS EQUAL TO THE VALUE OF THE
C     LOWER BOUND SOLUTION
C
79    Z=WNOT-SUM
      Y=W(I)
      YVEC(I)=Z/Y
      IF(SFLAG.NE.1)GO TO 2
      ZSOL=TOTAL+V(I)*YVEC(I)
      SINT=TOTAL+(WNOT-SUM)*V(I)/W(I)
      IF(SINT.EQ.XNOT)RETURN
      GO TO 6
2     Z=TOTAL+V(I)*YVEC(I)
      SINT=TOTAL+(WNOT-SUM)*V(I)/W(I)
6     IF(SINT.LE.XNOT)GO TO 16
C
C  IF THIS POINT HAS BEEN REACHED, THE LARGEST INTEGER
C     CONTAINED IN THE VALUE OF THE CURRENT NON-INTEGER
C     SOLUTION IS LARGER THAN THE VALUE OF THE LOWER BOUND
C     SOLUTION, AND SO THE FRACTIONAL VARIABLE IS ASSIGNED
C     A ZERO VALUE
C
      DO 18 J=1,N
      IF(YVEC(J).LT.1.AND.YVEC(J).GT.0)GO TO 19
18    CONTINUE
19    L=L+1
5     CONTINUE
      R(L)=J
      X(J)=0
      GO TO 10
      END



      SUBROUTINE STORE(TOTAL)
      IMPLICIT REAL(Y-Z,$),INTEGER(A-X)
      INTEGER*2 V,W,ORDER,X,SVECT,R,F,G,H
      COMMON V(100),W(100),N,X(100),ORDER(100),WNOT,
     *YVEC(100),SFLAG,XNOT,SVECT(100),L,R(100),F(100),
     *G(100)
C
C  IF THIS ROUTINE HAS BEEN CALLED, TOTAL IS THE VALUE OF AN
C     INTEGER SOLUTION
C     TOTAL WILL BE COMPARED TO XNOT, THE VALUE OF THE
C     CURRENT LOWER BOUND SOLUTION, AND THE SMALLER OF THE
C     TWO SOLUTIONS WILL BE REPLACED BY THE LARGER
C
      IF(TOTAL.LE.XNOT)RETURN
      XNOT=TOTAL
      DO 4 I=1,N
      SVECT(I)=YVEC(I)
4     CONTINUE
      RETURN
      END
```

APPENDIX II

Logic Diagram for the SOLVIT Subroutine Contained in Appendix I



INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library Naval Postgraduate School Monterey, California 93940	2
3.	Director, Systems Analysis Division (OP 96) Office of the Chief of Naval Operations Washington, D. C. 20350	1
4.	Department of the Army Civil Schools Branch, OPO, OPD Washington, D. C. 20315	1
5.	Professor Harold Greenberg Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
6.	Major Robert L. Hegerich 706 Fourth Avenue Fort Ord, California 93941	1
7.	Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
A Branch and Exclude Algorithm for the Knapsack Problem

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Thesis

5. AUTHOR(S) *(Last name, first name, initial)*
HEGERICH, Robert Lawrence, Major, United States Army

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1968	28	9

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
[redacted]

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Naval Postgraduate School
Monterey, California

13. ABSTRACT

A branch and exclude algorithm for the solution of the "knapsack problem", max $\sum_{i=1}^{N} v_i x_i$ where $\sum_{i=1}^{N} w_i x_i \leq W$ and $x_i = 0,1$, is presented which requires relatively small amounts of computer running time and core storage allocation. In addition, a branch and bound scheme is developed. The branch and exclude method is then compared to the branch and bound method and to a branch and bound method given by Kolesar [2]. Computational results are given.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Integer Programming						
Mathematical Programming						
Knapsack Problem						
Fly-away Kit Problem						